Popular Computing 46

January 1977 Volume 5 Number 1

$2.50

--by Donald E. Knuth Computer Science Department Stanford University Stanford, California 94305

Knuth

Are Toy Problems Useful?

The instructive "Dialogue" between Fred Gruenberger and Richard Hamming in Popular Computing 41 [4] raises several interesting questions. Hamming makes the following observations, among others:

> "I strongly feel that the entertainment approach to computers produces bad, long term results... Don't publish made up problems whose only point seems to be that it takes a machine to find the answer... Computing capacity is a wealth, and I deny that the ability to pay for machine time will always justify the expenditure of it even if you want to do it... A pied piper like you has not the right to lead children where they love going; you have some obligations to society for their time, attitudes, and the machine time you nominally control... A lot of your problems do not appear to be useful to anyone. I would have you pick problems that would appear to be useful to someone."

Gruenberger responds that he learned computing

> "slowly and painfully. I'm trying to shortcut the pain and time lapse for the youngsters. Just what would you have me feed them? Three-dimensional heat transfer problems? ... How to penetrate the time-sharing system so as to be able to wipe out everybody's files? ... I don't lead children where they love going, as you put it--far from it. If I let my kiddies alone, they'd play Star Trek all day, or write Tic-Tac-Toe programs ... I think that computing per se is a useful thing; learning it is difficult for most people, but ... there is no reason why the task cannot be made mildly entertaining and fun."

continued on page 3

Friedberg's Sequence

The following sequence:

N	S	N	S
1	5	11	5
2	17	12	577
3	37	13	677
4	5	14	5
5	101	15	17
6	5	16	5
7	197	17	13
8	257	18	1297
9	5	19	5
10	401	20	1601

has for its Nth term the smallest factor of $4N^2 + 1$ (Richard Friedberg, An Adventurer's Guide to Number Theory, McGraw-Hill, 1968).

When N is of the form $5K\pm1$, S is always 5. S cannot be 2 or 3.

The sum of the first 20 terms shown above is 5232. Further sums are:

at N	Sum
50	45490
75	195675
100	372036

Problem: What is the sum of Friedberg's sequence at N = 1000? ☐

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year to the above rates. For all other countries, add $3.50 per year to the above rates. Back issues $2.50 each. Copyright 1977 by POPULAR COMPUTING.

Editor: Audrey Gruenberger
Publisher: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald

Contributing editors: Richard Andree
William C. McGee
Thomas R. Parkin

Advertising Manager: Ken W. Sims
Art Director: John G. Scott
Business Manager: Ben Moore

These are important issues. Although I am by no means a competent philosopher, perhaps I will be able to make a few related remarks which may be useful to someone.

To set the scene, let us suppose a nationwide survey has just revealed that 95 per cent of all computer time is being spent in playing games like Star Trek, Space War, or chess, or in watching Conway's "Game of Life." Would this necessarily be a bad thing? (Suppose further that 4 of the remaining 5 per cent is being used by C.I.A. agents, doing what they consider to be "useful.") I suppose most people would be shocked only by the first statistic, since game-playing is usually considered a waste of time and money. Perhaps this is a vestige of Puritanism: We are admonished that it is immoral to have fun when other people are suffering. We should rather be doing something, something useful, so that ... so that ... well, so that we can have some enjoyment in future years when we retire.

Jeremy Bentham, the nineteenth-century founder of utilitarianism, did not actually hold such a view; he asked rather, "To what shall the character of utility be ascribed, if not to that which is a source of pleasure?" [1] There is also massive popular acceptance of recreational pastimes, at least in their "proper place"; people rarely question whether or not the quarterback of a pro football team has a useful occupation, although his function is merely to help win a game. Crossword puzzles and jigsaw puzzles are as popular as ever, and computer-controlled ping-pong games have swept the country; since 50 million people can't be wrong, fun must be a Good Thing.

Yet Hamming, and many other people who do not condemn football players for pursuing a useless career, would insist that computing must also be useful in a stronger sense; it should not be merely entertaining. Why this double standard? I don't believe the cost of computer time is the real reason, since so much money is spent on pure entertainment; I think the real reason is that computer scientists have discovered a better way to enjoy themselves with the machines, a Higher kind of fun, so it is distressing for them to see people wasting their time with lesser amusements. Once a person discovers high quality music or literature, he doesn't want to see masses of people listening to or reading trivial things, and the same holds for high quality computing. Thus, Hamming's point is that a program which does something useful is more satisfying than one which doesn't.

Of course, we can't expect everybody to agree on what is useful or interesting or beautiful. An amusing instance of this occurs in the recent mathematics literature: Let s(n) be the sum of all divisors of n less than n, so that n is a perfect number if and only if n = s(n), and two numbers m,n are amicable when s(m) = n and s(n) = m. In general one might study sequences such as n, s(n), s(s(n)), s(s(s(n))), etc.; and Richard Guy and Mike Williams of the University of Calgary began their article [5] about such sequences as follows:

"Bombieri [2] has commented on the interest of various problems in number theory, including those concerned with the iteration of arithmetic functions."

Few people are aware of the joke they are making here; for Bombieri's actual comment [2] was that these problems hold no interest at all!

Professor Guy was presumably interested enough in the subject to spend considerable time preparing his paper; and as editor of the Research Problems department of the American Math Monthly he is unlikely to be completely off the track. Yet Bombieri, one of the world's foremost number theorists, holds a completely different view. I don't think it's easy to decide who is more correct. From a historical standpoint, one must conclude that perfect numbers (at least) are interesting, since it was the study of perfect numbers which led to the extensive theory of Mersenne primes, and Mersenne primes have recently proved to be useful in computer arithmetic. On the other hand, I am inclined to feel that most of the interesting properties of $s(n)$ have already been found, and the remaining problems are of interest only insofar as they might encourage people to sharpen certain computational and analytical tools. At any rate Guy will probably never convince Bombieri that aliquot sequences are interesting, and Bombieri will probably never convince Guy of the opposite.

Granting that tastes differ, what sort of problems should young people be given in order to help them discover the higher levels of computer programming enjoyment? The Gruenberger-Hamming dialogue made me think back to how I learned programming. I had the disadvantage (?) of growing up before computing was taught in college, so I had to make up my own problems. I remember my first program quite well, it was to find the prime factors of a 10-digit number entered onto the computer console. As I recall, the first draft of that program involved about 50 instructions, but during a period of two weeks (while sitting at the machine console all night) I found and removed roughly 100 bugs. The final program, which I still have on file, was 137 words long.

My second program was radix conversion, taking a decimal number and converting it to another base $b<10$, yet displaying the answer in the lights of our decimal computer. I don't recall much about that program, so I guess it wasn't especially interesting. My third program was written in an interpretive system, over 300 lines of numeric three-address code; it found the roots of fifth degree polynomials, by finding one real root and then plugging into a quartic formula I had found in a book. That program didn't excite me at all because it worked correctly the first time I ran it. (At least I thought it did.)

My fourth program is the one I remember most of all, since I spent two months writing and debugging it. Fred Gruenberger probably will say I learned programming in spite of this experience, because I chose to work on one of the few things he especially denounced! Yes, my fourth program played Tic-Tac-Toe.

It did this in three different ways: (a) By following a predetermined built-in forcing strategy, (b) by examining all possible moves and choosing the best, and (c) learning by past experience what was good and what was bad. I was especially excited by the task of designing the program so that (c) could play against (a), (b) or (c). Incidentally, when (c) played (c) it was like blind leading the blind; after about 350 games the two sides learned to draw against each other by playing safely.

Well, I don't think I wasted computer time that summer, since I learned a lot. After the Tic-Tac-Toe experience I began to read other people's programs, especially Stan Poley's assembler and Alan Perlis's compiler, and I got hooked on software; this turned out to be even more fun, but I wouldn't have believed it when I first started. The toy problems were instrumental in helping me to think algorithmically.

That was 19 years ago. By now I have learned somewhat more about programming, but for some reason I haven't lost all interest in toy problems. I'd like to discuss two of them which recently caught my fancy, since these specific examples may shed some light on the general question of problem design.

The first problem is one that Bob Floyd posed to the beginning graduate students in our Ph.D. program at Stanford during the fall of 1972:

"The numbers $\sqrt{1}, \sqrt{2}, \ldots, \sqrt{50}$ are to be partitioned into two parts whose sum is nearly equal; find the best such partition you can, using less than 10 seconds of computer time."

For example, it turns out to be possible to find the best partition of the smaller set of numbers $\sqrt{1}, \sqrt{2}, \ldots, \sqrt{30}$, after only about one second of computer time; the answer is

$$\sqrt{2}+\sqrt{6}+\sqrt{9}+\sqrt{11}+\sqrt{12}+\sqrt{13}+\sqrt{14}+\sqrt{21}+\sqrt{23}+\sqrt{24}+\sqrt{25}+\sqrt{26}+\sqrt{27}+\sqrt{30}$$

$$\approx 56.04142258807335185163208 26\ ;$$

$$\sqrt{1}+\sqrt{3}+\sqrt{4}+\sqrt{5}+\sqrt{7}+\sqrt{8}+\sqrt{10}+\sqrt{15}+\sqrt{16}+\sqrt{17}+\sqrt{18}+\sqrt{19}+\sqrt{20}+\sqrt{22}+\sqrt{28}+\sqrt{29}$$

$$\approx 56.0414226276195573033211496\ .$$

Note that the two sums agree to nine significant digits. The problem with 50 instead of 30 is much more difficult, and it appears hopeless to find an absolutely optimum partition in only 10 seconds. This is one of the beautiful features of Floyd's problem, since it allows for a friendly competition between the members of the class (with a tie score very unlikely), and especially because it makes the problem typical of real life situations. We are often confronted with problems that cannot be solved exactly at reasonable cost, so we must do the best we can under finite limitations. I have tried to explain in [8] why such limitations are valuable. The time restriction encourages us to think, not merely to compute!

While solving Floyd's problem, a student will be developing a variety of important skills. He will be dealing especially with questions of how to handle large quantities of data and how to estimate computation time; the fact that square roots of numbers are involved is actually of little relevance, compared to these other issues which are of such fundamental importance in computer science. Floyd's problem is simply stated, so that a student will not get bogged down in the sea of minutiae which often is present in real-life situations; the paradigms of computer science are learned more easily when the problem is not cluttered up by extraneous details. On the other hand, the fact that square roots are involved does add just enough anomalies to keep a student from confining himself to an approach that is "too pure"; for example, one thing that might be considered when solving Floyd's problem is the fact that $\sqrt{3} + \sqrt{12} = \sqrt{27}$.

For these reasons, I believe that Floyd's problem is excellent, even though it fails Hamming's criterion of "appearing to be useful to someone." The best bipartition of

$$\left\{\sqrt{1}, \sqrt{2}, \ldots, \sqrt{50}\right\}$$

is hardly likely to be of any use whatever to anybody, yet one learns important skills by looking for it. After all, the purpose of computing is insight, not numbers, and one good way to demonstrate this is to take a situation where the numbers are clearly less important than the insights gained. It will not be especially interesting to know the exact value of the best partition; it is much more interesting to devise a way to compute a good approximation in less than 10 seconds.

I suggest that the reader of this article will find it very instructive to spend an hour thinking about how to solve Floyd's problem. Since I greatly enjoyed working on it, I can't restrain myself from writing down my own solution; but it would spoil your fun if I explained it now, so I have asked the editor to save that part of my article for the next issue of Popular Computing.

The second problem I would like to mention is perhaps most interesting because it was condemned by the great G. H. Hardy. In his fascinating little booklet, A Mathematician's Apology [6], Hardy explains the difference between what he calls 'real' mathematics and 'trivial' mathematics, and points out that this distinction is better than the conventional dichotomy between 'pure' and 'applied.' (Hardy's 'real' mathematics--the kind which has permanent aesthetic value--does however tend to be mostly pure, and what he calls 'trivial' mathematics--the kind which is usually taught in schools--tends to be mostly applied.) He closes with some autobiographical remarks:

> "I have never done anything 'useful.' No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world. ... I have just one chance, ... that I may be judged to have created something worth creating."

Thus we get some idea of his philosophy.

In Section 15 of Hardy's book he begins to explain what a 'serious' or 'significant' mathematical theorem is, by giving two examples of insignificant ones. His second example is the fact that there are just four numbers (greater than 1) equal to the sums of the cubes of their digits, namely

$$153 = 1^3 + 5^3 + 3^3$$

$$370 = 3^3 + 7^3 + 0^3$$

$$371 = 3^3 + 7^3 + 1^3$$

$$407 = 4^3 + 0^3 + 7^3.$$

He calls this an odd fact, "very suitable for puzzle columns and likely to amuse amateurs, ... [but the proof is] neither difficult nor interesting--merely a little tiresome."

In other words, Hardy considered this problem to be worse than useless. Yet I recently devoted a fair amount of time to discovering such odd facts as these:

$$564240140138 = 5^{13} + 6^{13} + 4^{13} + 2^{13} + 4^{13} + 0^{13} + 1^{13} + 4^{13} + 0^{13} + 1^{13} + 3^{13} + 8^{13}$$

$$94204591914 = 9^{11} + 4^{11} + 2^{11} + 0^{11} + 4^{11} + 5^{11} + 9^{11} + 1^{11} + 9^{11} + 1^{11} + 4^{11}.$$

There are no solutions with exponent 12, and the unique solution for exponent 10 is 4679307774; there are eight solutions with exponent 11. Although I completely agree with Hardy that the results are mathematically trivial, I cannot agree that the proof (in this case, by computer of course) was uninteresting; in order to avoid considering a large number of cases I had to devise a technique which might be called "double backtracking." Thus I found the problem quite instructive, although I certainly don't consider it the most important thing I have ever done.

Hardy himself made the following remarks in another part of his book:

> "One rather curious conclusion emerges, that pure mathematics is distinctly more useful than applied. ... For what is useful above all is <u>technique</u>, and mathematical technique is taught mainly through pure mathematics."

And that is why I liked this useless problem: computational technique is taught mainly through "pure" computer science.

Most of the exercises in calculus books are strictly academic, without any indication of how they might connect to the real world; and my point is that there is good reason for this, because actual applications are usually full of exceptions to the rules and other things which distract from the main principles a student is presently learning. Furthermore it is usually necessary to have a fair amount of background knowledge in a particular application area before a "useful" problem can properly be appreciated; with a diverse group of students, suitable real-world applications are rare, but when the students have a common background it may well be that three-dimensional heat transfer problems are the most exciting and instructive.

Of course we should not make the mistake of saying that pure problems are the only good ones; I'm arguing that they have a well-deserved place in a person's education, but one's education is woefully inadequate if it is based purely on such problems. We should remember Gibbon's criticism of the Greeks: "The whole of life was spent studying the art of reasoning, and never in actually reasoning." [3]

Thus it is extremely important to learn how to connect theory with practice; to learn that theory builds up one's conceptual apparatus but that one rarely actually uses a concept in the way he originally learns it. The limitations of a formalism must be understood clearly; I have written elsewhere about some of the more notorious abuses of computer science theory [7].

We need to know how to deal with complicated systems of rules and exceptions. That is why, for example, I have included a detailed description of a real-life elevator system in the first volume of my books on programming, and why I intend to discuss COBOL's "picture clauses" at some length in volume five.

One of the most difficult tasks I had to face when writing the third volume was the preparation of Section 5.4.6, "Practical Considerations for Tape Merging"; for it deals with precisely this question of how theory combines with practice, a question which is almost never discussed in the literature. I wanted to explain how real magnetic tape units operate; and this led to the complexities of interblock gaps, start/stop time, dual rewind mechanisms, read/write/compute overlap, and the finiteness of tape, besides the problems of allocating an appropriate amount of buffer space and waiting for the computer operator to change a tape. I found that most of the literature's "improved" methods, which looked so good on paper, actually performed poorly in practice, compared to the simpler methods of the 1940's; I had to make substantial changes to these "improved" methods before they really represented any improvement. It was also necessary to bend the theoretical formulas so that they would appropriately model the real-world assumptions without becoming too detailed.

Since I feel that Section 5.4.6 teaches some very important lessons, I guess I'm proud of how it came out, in spite of its imperfections. Therefore I have been disappointed to find that many of my students aren't especially interested in this section; their reason is that tapes are being replaced by disks nowadays. In other words, my discussion of "practical considerations" has become largely a toy problem, since the technology has changed! Although everybody would have loved this section in 1966, many of the students would rather not read it at all in 1976; they fail to realize that technology will have changed again by 1986, when they will want to apply their education. The important thing for them to learn now is the methodology of combining theory with practice, <u>not</u> the details of 1976 practice; the former will last a lifetime, but the latter will soon be obsolete. Thus I believe Section 5.4.6 should continue to be interesting and relevant even after the last magnetic tape unit has vanished from the earth.

Perhaps I can summarize all of these remarks by saying that <u>methods are more important than facts</u>. The educational value of a problem given to a student depends mostly on how often the thought processes he invokes to solve it will be helpful to him in later situations; it has little to do with how useful the answer to the problem may be.

On the other hand, a good problem must also motivate the student; he should be interested in seeing the answer. Since students differ so greatly, I can't expect everyone to like the problems that please me; Floyd's problem and the powers-of-digits problem will probably leave many people cold, especially those who are not already somewhat fluent in algorithmic thinking, and I don't expect that every student will be excited about tape sorting, in spite of my enthusiastic endorsement. A wide variety of problems is certainly needed; but "usefulness" should not be the main consideration.

References

[1] Jeremy Bentham, The Rationale of Reward, Book 3, Chapter 1. (First published in 1811.)

[2] Felix E. Browder, ed., "Problems of present day mathematics," in Mathematical Developments Arising from Hilbert's Problems, Proc. Symp. Pure Math. 28 (Amer. Math. Society, 1976), 35-80 Section 2A.

[3] Edward Gibbon, Mémoires de l'Académie des Inscriptions 6 (1729), 151; quoted in Giuseppe Giarrizzo, "Gibbon's other historical interests," Daedalus 105 (1976), 49-62.

[4] Fred Gruenberger and Richard Hamming, "Dialogue," Popular Computing 4, 8 (August, 1976), 3-7.

[5] Richard K. Guy and M. R. Williams, "Aliquot sequences near 10^{12}," Proc. Fourth Manitoba Conf. Numer. Math. (1974), 387-406.

[6] G. H. Hardy, A Mathematician's Apology (Cambridge University Press, 1940). The 1967 edition of this book includes an informative introduction by C. P. Snow.

[7] Donald E. Knuth, "The dangers of computer science theory," in Logic, Methodology, and Philosophy of Science, 4, ed. by P. Suppes et al. (North-Holland, 1973), 189-195.

[8] Donald E. Knuth, "Computer programming as an art," Comm. ACM 17 (1974), 667-673, especially the section called "Less Facilities: More Enjoyment."

[9] Benjamin L. Schwartz, "Self-generating integers," Mathematics Magazine 46 (1973), 158-160.

In issue No. 16, page 15, the following problem (due to Kraitchik) appeared:

Divide the number 10 into two parts,

$$X + Y = 10$$

such that the product of each part plus its square root is 23; that is,

$$(X + \sqrt{X})(Y + \sqrt{Y}) = 23$$

For Kraitchik's problem, the result, as calculated by Herman P. Robinson, is:

.93701 03835 13028 71730 39274 81437 28161 64348 87554...

Suppose we change the problem slightly:

$$\left.\begin{aligned} X + Y &= 10 \\ (X + \sqrt{X})(Y + \sqrt{Y}) &= 40 \end{aligned}\right]$$

and for this system X has the value 2.302314951...

Now consider the system:

$$\left.\begin{aligned} X + Y &= 20 \\ (X + \sqrt{X})(Y + \sqrt{Y}) &= 80 \end{aligned}\right]$$

we will not get the same result, even though it appears that the problem has simply been "scaled up" by a factor of two. If we take the more general problem:

$$\left.\begin{aligned} X + Y &= N \\ (X + \sqrt{X})(Y + \sqrt{Y}) &= 4N \end{aligned}\right]$$

the value for X will be around 2 for all values of N equal to or greater than 8, but in a rather peculiar way. At N = 8, X is nearly 2.5. At the other extreme, when N is very large, X is around 2.43.

Problem: Find the value of N for which X is the smallest (somewhere around N = 24.77).

This problem lends itself to <u>double</u> <u>bracketing</u>: For each value of N, the bracketing process (see issues 35, 39, and 44 for discussion of this process) can be used to find X; then the whole procedure can be bracketed to find N.

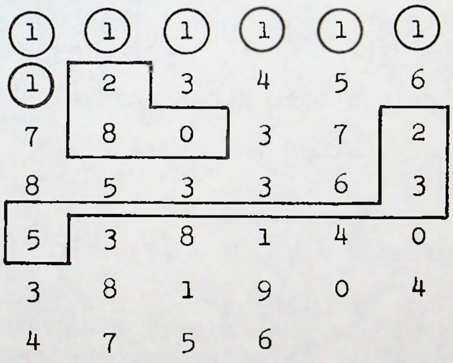
PROBLEM **156**

☐

CONTEST 9 RESULTS

K-COLUMN FIBONACCI

We will review the K-Column Fibonacci Problem in terms of case k = 6. The results for cases k = 1, 2, 3, 4, 5 were given with the problem statement, in issue No. 40. For k = 6, we have:



1	1	1	1	1	1
1	2	3	4	5	6
7	8	0	3	7	2
8	5	3	3	6	3
5	3	8	1	4	0
3	8	1	9	0	4
4	7	5	6		

The starting configuration is the set of circled numbers. Each subsequent number is found by adding the numbers in a vertical pair (reduced modulo 10) and proceeding in a rotating cycle, as shown by the enclosed numbers. The problem was to determine how many rows of the pattern are needed to return to the starting array.

For k = 6, the result is 2480437 lines, as determined by contest winner Sam Wagner, Bluffton College, Bluffton, Ohio, who also obtained the results:

k = 7	2232
k = 8	7128815

➧

Mr. Wagner, using results reported in the literature, proceeded as follows:

$w_1 = 1,\ w_i = 1,\ \ldots,\ w_{k+1} = 1$ where k is greater than 1 and an integer. k is the number of columns in the array.

$w_n = w_{n-1} + w_{n-k-1}$ for n greater than k, n integral.

Let $P_k(m)$ = period of sequence mod(m)

Find $P_k(2)$ and $P_k(5)$

Then $P_k(10) = \mathrm{LCM}(P_k(2), P_k(5))$

To get $P_k(S)$, we computed the terms mod 5 until we obtained (k+1) consecutive ones. We then took the term preceeding the first term in the string of ones:

$$w_i = w_{n-k-1}$$

n is the term number of the last of the ones in the string of ones. Then

$$P_k(5) = \mathrm{LCM}(w_i, k)/k$$

$$P_k(10) = \mathrm{LCM}(P_k(2), P_k(5))$$

The complete tabulation of results is as follows:

k	P(2)	P(5)	P(10)
1	3	20	60
2	7	31	217
3	5	104	520
4	21	6	42
5	63	3124	196812
6	127	19531	2480437
7	9	2232	2232
8	73	97655	7128815

Further results for k = 9, 10,... are solicited. □

Comparison of a Strong BASIC with a Standard Fortran

--by James L. Boettler
Talladega College
Talladega, Alabama

I feel that a strong BASIC language is more valuable than a standard Fortran language, in a teaching environment, because a program written in the strong BASIC:

1. Requires less advanced syntax.
2. Is usually shorter.
3. Is easier to read.
4. Is easier to write.

Below are three sample problems that might be assigned in an elementary computing course, along with solutions in two languages: PDP-11 RSTS/E BASIC, and IBM 1130 Fortran (not tested).

1. Write a program to print the product of 5 and 7.

<u>BASIC</u>

```
10 PRINT 5*7
```

<u>Fortran</u>

```
      K = 5*7
      WRITE (3,20) K
20    FØRMAT (1X, I2)
      CALL EXIT
      END
```

2. Write a program to print the Fibonacci numbers, 5 or more numbers per line, up to around 10,000.

<u>BASIC</u>

```
10 A, B = 1 \ C = 2
20 PRINT A, B, C,
30 A=B+C \B=A+C \C=A+B
40 IF A <10000 THEN 20
50 END
```

<u>Fortran</u>

```
      DIMENSIØN K(20)
      K(1) = 1
      K(2) = 1
      DØ 50 J = 3, 20, 1
         K(J) = K(J-2)+K(J-1)
50    CØNTINUE
      WRITE (3,60) K
60    FØRMAT (1X, 5I14)
      CALL EXIT
      END
```

3. Write a program which searches a string of characters for the substring "AND". If found, print the column number the substring starts in. Repeat program until aborted.

```
    BASIC                             Fortran

100 A$ = "AND"                   C.......L=string, IA=substring,
200 INPUT LINE L$                C.......N=# of letters
210 L = INSTR(1, L$, A$)
220 IF L > O THEN PRINT L                DIMENSIØN L(80), IA(3)
230 GØTØ 200                             DATA IA, N / "A","N","D", 3 /
990 END                             20   READ (2,30) L
                                    30   FØRMAT (80A1)
                                         K9 = 81 - N
                                         DØ 100 K = 1, K9
                                            IP = 0
                                            DØ 80 J = 1, N
                                               M = K+J-1
                                               IF (L(M)-IA(J)) 80,60,80
                                    60         IP = IP + 1
                                               IF (IP-N) 80, 200, 80
                                    80      CØNTINUE
                                    100  CØNTINUE
                                         GØTØ 20
                                    200  WRITE (3,210) K
                                    210  FØRMAT (1X, I5)
                                         GØTØ 20
                                         END
```

The first problem is one that is often assigned the first week of computer classes, before the student knows anything about computing. The BASIC version is simplicity itself, and requires very little explanation to the student. But a teacher may need a full period or more to explain the details of the Fortran version. Some questions that students might raise concerning the Fortran version are:

a. Is 5*7 = K legal?
b. Why the numbers in the WRITE statement?
c. Why not use WRITE (3,20) 5*7 and eliminate line 1?
d. Can line 1 appear anywhere?
e. Why the FØRMAT line? Can it go after the END?
f. Must we use a separate card for each line?
g. Why not use FØRMAT (1X, I2) 5*7 and eliminate the lines before it?
h. I forgot the 1X; why did I get an answer of 2 instead of 12?

You could probably add dozens of other questions that students might raise. But they'll ask very few questions about the BASIC version.

The second problem is one that might be assigned near the middle of a computer language course. The Fortran version requires that the numbers be saved, in order to print several answers on the same line. Thus, a more advanced technique--arrays--must be used (or, more variables and statements could be used). Students tend to be baffled by the lack of an index on the array K in the WRITE statement. Some students get confused because they think that line 60 can yield only 5 values. The BASIC version has a logic problem in line 30 which students have to think about before they understand it. But everything else is clear in the BASIC version.

The third program is one that might be assigned near the end of a course. The BASIC version takes advantage of the in-string search function, INSTR (called WRD in Hewlett-Packard 3000 BASIC; EXT in GE BASIC; POS in Dartmouth BASIC, and in PDP-8 OS8 BASIC--other names probably exist, but the idea is the same). Once that line is grasped, the rest of the program is simple. But the Fortran version is difficult to follow, because Fortran has no string variables or functions. The BASIC version has a single GOTO loop. But the Fortran version requires loops nested three deep, requires arrays, and requires that the programmer look at characters as though they were numbers.

BASIC was created for simplicity, with many defaults provided. Fortran has lots of arbitrariness built into it, which confuses students. A major stumbling block is the FORMAT statement, with numerous seemingly arbitrary codes (I, F, E, A, H, X, /, G) and confusing line printer controls (blank, 0, 1, +). Subscripts must be integer; only 7 forms are allowed; subscripts themselves can't be subscripted. DO loops always execute at least once, which means you must precede the DO loops with IF statements if the loops shouldn't be executed. The codes for the relational operators are abbreviations, rather than the standard symbols used in mathematics courses. No string variables or functions are provided, so work with strings requires the programmer to resort to trickery, as shown in example 3. Fortran does not have IF-THEN-ELSE statements, or multiple statements per line.

Some BASICs have all the power necessary to do anything that a Fortran or assembly-language program can do. The drawbacks to BASIC are minor: short variable names; interpretation (usually) rather than compilation; it is not yet used commercially to any extent--but these drawbacks are disappearing.

TROUBLES!

The problem below comes from contributing editor Professor Richard Andree.

The following program in WATFIV version of Fortran was intended to make a table of K and K! for values of K from 5 to 26.

```
          DØ 13  K = 5,26
            NIF = 1
            DØ 12  L = 2,K
12         NIF = NIF*L
          WRITE (6,112) L, NIF
112       FØRMAT(1X,I3,I16)
13         CØNTINUE
          STØP
          END
```

When the program was run on an IBM 370/158, the following output was obtained.

```
  6             120
  7             720
  8            5040
  9           40320
 10          362880
 11         3628800
 12        39916800
 13       479001600
 14      1932053504
 15      1278945280
 16      2004310016
 17      2004189184
 18      -288522240
 19      -898433024
 20       109641728
 21     -2102132736
 22     -1195114496
 23      -522715136
 24       862453760
 25      -775946240
 26      2076180480
 27     -1853882368
```

PROBLEM 157

The diagnostics state:

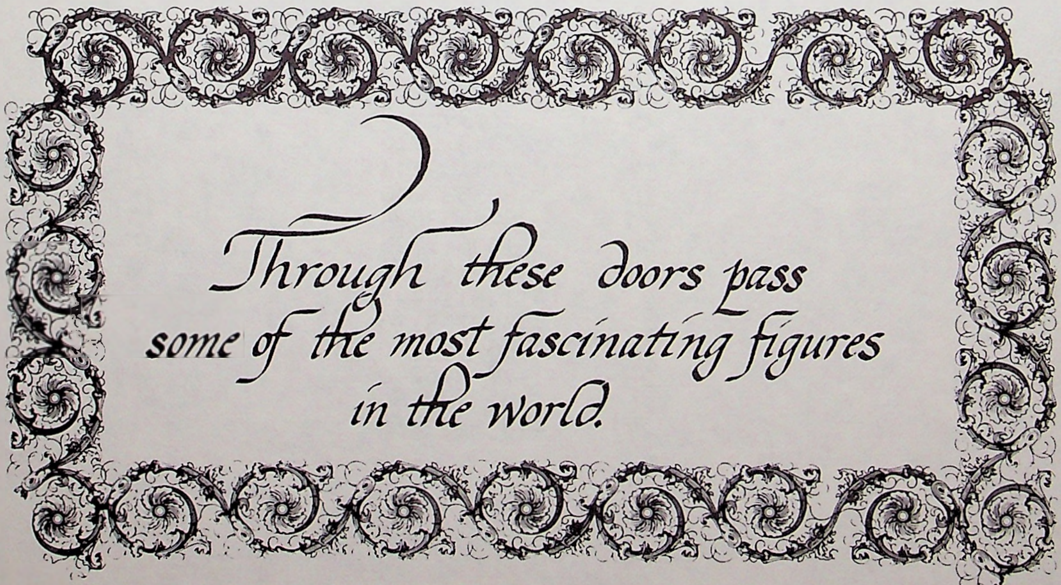
```
NUMBER ØF ERRØRS = 0

NUMBER ØF WARNINGS = 0

NUMBER ØF EXTENSIØNS = 0
```

Nevertheless, there are at least three trouble spots that merit debugging:

1. The numbers opposite 6, 7, 8, etc., seem to be 5!, 6!, 7!, etc., judging by the first few lines, so the K column seems to be one digit high. It's easy to fix, but _why_ did it come out that way?

2. The number opposite 13 is 479001600, which is the correct value for 12! However, the number opposite 14 is 1932053504, whereas the correct value for 13! is 6227020800 according to the table of factorials that appeared in issue No. 2 of POPULAR COMPUTING, and the latter value agrees with 13! as worked out on my pocket calculator. What happened in the computer results?

3. The values opposite 15 to 27, which might be expected to correspond to 14! to 26! all seem garbled, but what bothers me most is how on earth some of them came out _negative_. Should I have anticipated this? What happened to the low order zeros?



Chirp, Chirp

The rate at which crickets chirp is a function of the temperature, a fact first noted by the astronomer Otto Struve. The following table shows some empirical data between X (chirps per minute) and Y (temperature Fahrenheit):

X	Y	X	Y
56	55	80	66
60	57	84	68
67	60	87	70
70	61	92	71
73	63	94	72
85	64	100	76
75	65	104	77
78	65	96	80

We seek a functional relationship, showing Y as a function of X. The first thing to do is to plot the given data, in order to make a judgement as to what sort of function to seek. The graph will reveal two things:

a) The data appears to be linear; that is, a curve of the form Y = AX + B should provide a good fit to the data.

b) While most of the data points lie along a line, two of them (85, 64) and (96, 80), seem to be off. This is empirical data--someone stood outdoors with a thermometer and a watch and counted, so small errors in the data are expected and natural. Large errors could be caused by many factors:

a) The recorded temperature should be the temperature for <u>that</u> cricket, and he may be in a colder or hotter place than the observer.

b) We are trying to express a law of cricket chirping. Any given cricket may not choose to conform to the law, or may be of a different breed, or may be ill, or stupid.

In any event, on the assumption that a linear relation is appropriate, and that the given data is all we have to go on, then the normal equations:

$$\left|\begin{aligned} &A\Sigma X^2 + B\Sigma X = \Sigma XY \\ &A\Sigma X + BN = \Sigma Y \end{aligned}\right.$$

(N = 16 is the number of data points) furnish the means of finding the values of A and B. Do so.

(Taken, with modifications, from <u>Computing With the BASIC Language</u>, Canfield Press, 1972.)

Log 46	1.662757831681574074081516006975682576465700915798205
ln 46	3.828641396489095000223984953268372686517880449200691
$\sqrt{46}$	6.782329983125268139064556326625969105195748323923288
$\sqrt[3]{46}$	3.583047871015946485386963674580666272154820593095635
$\sqrt[10]{46}$	1.466478780211680893467357764358389314914267343202387
$\sqrt[100]{46}$	1.039028782618528851295177515828399745259972058698971
e^{46}	94961194206024488745.1336491171183231018171589210799878504381651795835627726037033002
π^{46}	73942502370765147684186.5841458791081207681603345761705902994458396381675555172701
$\tan^{-1}$ 46	1.549060619953103953323653838755778837378154641910368

An ad for the SR4190R (CBM Commodore, U.K., Ltd.) which sells in Britain for about \$83.50 has this:

> THE ACCURACY TEST: Before you buy a scientific check the accuracy with this simple test:
>
> $29 \sin \cos \tan \sqrt{x} \ln e^x x^2 \tan^{-1} \cos^{-1} \sin^{-1} = ?$

The 29 is degrees, of course, since no machine will invert the trig functions on 29 radians. The SR-52 of Texas Instruments gives this result: 29.00001537. Various Hewlett-Packard machines return values around 29.00xxxx, inasmuch as they carry their calculations to only 10 significant digits. The CBM ad fails to state just what the result <u>is</u> on their machine.